A control-theory approach for cluster autonomic management: maximizing usage while avoiding overload

Agustín Gabriel Yabo*, Bogdan Robu†, Olivier Richard*, Bruno Bzeznik*, and Éric Rutten*
*Univ. Grenoble Alpes, Inria, CNRS, LIG, F-38000 Grenoble France
†Univ. Grenoble Alpes, CNRS, Grenoble INP, GIPSA-lab, F-38000 Grenoble France
Email: {firstname.lastname}@inria.fr, bogdan.robu@gipsa-lab.grenoble-inp.fr

***Abstract*— Cloud and HPC (High-Performance Computing) systems have increasingly become more varying in their behavior, in particular in aspects such as performance and power consumption, and the fact that they are becoming less predictable demands more runtime management. In this work, we describe results addressing autonomic administration in HPC systems for scientific workflows management through a control theoretical approach. We propose a model described by parameters related to the key aspects of the infrastructure thus achieving a deterministic dynamical representation that covers the diverse and time-varying behaviors of the real computing system. Later, we propose a model-predictive control loop to achieve two different objectives: maximize cluster utilization by best-effort jobs and control the file server's load in the presence of external disturbances. The accuracy of the prediction relies on a parameter estimation scheme based on the EKF (Extended Kalman Filter) to adjust the predictive-model to the real system, making the approach adaptive to parametric variations in the infrastructure. The closed loop strategy shows performance improvement and consequently a reduction in the total computation time. The problem is addressed in a general way, to allow the implementation on similar HPC platforms, as well as scalability to different infrastructures.**

Keywords : Control of Computing Systems, High-Performance Computing, Software Systems, Model-Predictive Control

I. Introduction

A. Control Theory for High Performance Computing

Cloud and HPC (High-Performance Computing) systems have increasingly become more varying in their behavior, particularly in aspects such as performance and power consumption. The fact that they are are becoming less predictable demands more runtime management, requiring frequent human intervention. Coping with such phenomena must be done online based upon measurements performed during execution time. This can be done in response to monitored sensors of the systems, by analysis of this data and utilization of the results in order to trigger appropriate system-level or application-level reconfiguration mechanisms. There is not a single way to perform this reconfiguration, one of them being the development of feedback loops (see for example [1]) which in the domain of Computer Science are the object of Autonomic Computing [2].

Control theory has been applied in High-performance Computing rather scarcely therefore it is still in need for basic research [3], [4], [5]. Most previous approaches propose rather simple linear models that, to our best understanding, lack the ability to reflect several non-linear behaviors and variable constraints of the real system, since their accuracy is limited to a specific *region of operation*, failing to ensure the desired performance and stability objectives for all possible situations.

In [6], such a predictable HPC system was designed in order to control each job's waiting time and running time. Moreover, they proposed an admission control scheme by deciding to accept or reject incoming jobs in terms of their deadlines. To achieve this, they model each job's progress with a first order linear difference equation. However, the proposed solution assumes each job's progress can be measured by the algorithm, which is not always true in HPC environments.

In [7], a model predictive admission control approach is proposed for performing real-time scheduling in HPC clusters. They divided workflows into 2 categories (called *Data* and *Design*), with different time duration each. Thus, they assume to count on *a priori* knowledge of the job's duration (that should be provided by the user), leading to an *ad-hoc* solution that is not robust to variations in the analyzed workflows, and requires *offline* analysis of the system.

B. Contributions

This work draws from a preliminary one [8] where we approximated the cluster dynamics as a second order system, considering its controlled variable as the number of jobs in the cluster's waiting queue. Then a PI controller was implemented in order to track a reference number of jobs by regulating the amount of jobs submitted to the cluster.

While focusing on avoiding cluster overload our previous work was very simplistic, needed strong preliminary hypothesis and good system comprehension for implementing control. Moreover, it didn't consider another major cause of cluster overload which is filesystem overload and which depends directly on the amount of running jobs in the cluster.

In this work, we present a model that comprises both the waiting queue as well as the running jobs in the

cluster, based on fluid modeling theory [9]. Along with this model, we investigate the dynamics of a fileserver's load average, and propose a scheme to measure the impact of scientific computing workflows onto it. Later on, we implement a model-predictive control loop that relies on an EKF (Extended Kalman Filter) for online parameter estimation to provide an accurate prediction and robustness to variations in the workflow. Ultimately, the scheme deals with two objectives: a) maximize cluster usage, and b) regulate the fileserver's load. While the solution is presented in a general way, we focused on a scheme that have not received much attention from the control community: scientific workflows management. In this context, there are a number of assumptions that can greatly simplify the general HPC scheduling problem (e.g. specific probability distributions for execution times, rather uniform resource consumption, etc.). Last but not least, the solution is non-intrusive, it doesn't interfere with the scheduling process of the jobs.

II. Background

A. Experimental setup

All the experiments in this paper are done on the CIMENT HPC production cluster detailed below.

1) CIMENT center: the CIMENT center is one of the most powerful High Performance Computing (HPC) tier-2 centers in France. It provides HPC resources to academic research communities from a wide range of disciplines: Biology and Health, Chemistry, Environment and Climate, Numerical Physics, Earth and Planetary Sciences, and Distributed Computing [10].

2) CiGri middleware: CiGri[1] is a simple, lightweight, scalable and fault tolerant grid system designed to exploit the unused resources of a set of computing clusters. The software was developed at Université Grenoble Alpes and actively used on the production clusters of the CIMENT center, providing users the possibility to launch and manage large scale scientific computations. It interacts with the computing clusters through OAR [11], a modular batch scheduler for HPC clusters. The platform supports only one specific form of job submission: the bag-of-tasks workload, composed of a set of independent parametric jobs, which represent the lowest level of granularity in the system.

The nature of this specific kind of workload arises from the need of running large-scale scientific computations with different parameters, by means of grid computing infrastructures. As previously studied from historical data from CiGri database [8], these jobs show small variability in their execution time, with coefficients of variability [12] that ranges from 10% to 30%, indicating minor dispersion.

3) Utilization policy: The computing grid supports campaign launching through two different submission mechanisms:

[1]http://ciment.ujf-grenoble.fr/cigri/dokuwiki/doku.php

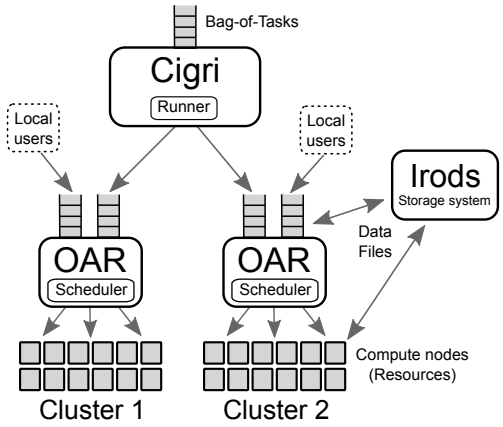


Fig. 1: The system global architecture.

- Campaign submission through CiGri: the user submits a campaign, which yields a parametric bag-of-tasks in the CiGri server (Figure 1), and the *Runner* module subsequently sends its jobs to a dedicated waiting queue in each OAR cluster. This way, CiGri balances the amount of jobs submitted onto all the clusters.
- Direct job submission through OAR: the user decides the cluster on which the job is going to run, and submits it through its *Resource Job Management System* to the *Local users* queue in the cluster (Figure 1). In this case, the user is not constrained to run only parametric jobs.

This latter makes the system essentially different from other, extensively explored, problems in IT infrastructure. Such is the case of Cloud systems [13], [14], [15], where there is no such differentiation in the grid between CiGri and user jobs. MapReduce is a more distributed parallel-oriented framework that targets large parallelizable problems, while CiGri focuses on independent parametric tasks. Another key difference is that, even though in both environments the resources are shared, in CiGri they are treated as a varying amount of dedicated resources, which poses a different challenge in terms of management.

B. Feedback loop objectives

In order to exploit idle resources in a transparent manner, CiGri introduces the concept of *best-effort* jobs into OAR, which are treated as 0-priority tasks by the scheduler. This means that, if during their execution in a specific resource, this latter is requested by a local cluster user, the job is killed by the local batch scheduler and later resubmitted according to specific fault-treatment mechanisms [16]. Hence, the challenge of CiGri is to guarantee the complete execution of this scientific computations in spite of resources' volatility, in the most efficient way. This implies maximizing the usage of idle resources by best-effort jobs, without overloading any component of the infrastructure.

The two main issues in the CiGri environment that are tackled throughout this work are:

- **Resource under-utilization**: the algorithm cyclically submits a number of jobs to the cluster (which increases at every iteration) and waits for the completion of all submitted jobs. In practice, this behavior yields situations where there are idle resources in spite of the existence of remaining jobs in the bag-of-tasks, which translates into the cluster being under-used (or not used at all).
- **Fileserver overload**: every computing job consumes storage resources, whether for reading input files from a server, or for storing the output of the script. The simplest I/O task can greatly affect fileserver's performance when running hundreds of simultaneous jobs. This makes storage a major challenge in parallel computing infrastructures, and a limiting factor in the performance of the grid [17]. In the CiGri infrastructure, fileserver overloading is a common problem that requires a human operator to intervene in order to remedy the situation.

The main objective of this paper is to improve the job submission mechanism implemented in CiGri. The way CiGri handles job submission to the cluster is a major factor in the overall performance and, thus, maximizing the exploitation of idle resources requires an improved strategy in this regard.

III. Modeling and control

A. Queue modeling

1) Overview: In this section, we model the behavior of a single cluster managed by an OAR scheduler (Figure 2). The proposed model comprises the dedicated waiting queue for *best-effort* jobs in the scheduler (scheduler's left queue in Figure 2) of length q, considering the jobs sent by the runner as the input u. The number of jobs taken from the waiting queue and allocated into resources by the scheduler is denoted as b (for buffer), whereas the number of *best-effort* running jobs in the cluster is represented by r.

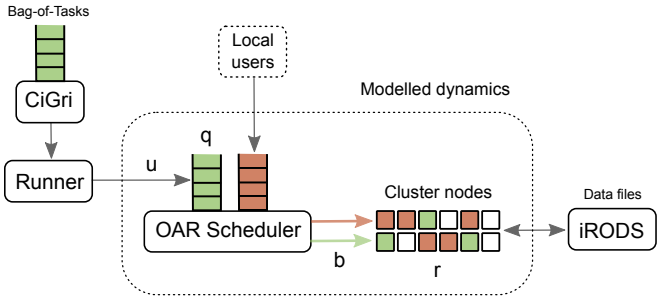


Fig. 2: Scheme of a CiGri environment single cluster.

The model is characterized by the following parameters:

- **Maximum number of jobs allowed** r_{max}: the scheduler can allocate a maximum number of jobs r_{max} into the cluster depending on resource availability[2]. The impact of users' jobs (not managed by CiGri) is represented in the model through the variations of this parameter.
- **Job allocation's buffer** b: the job allocation process is entirely managed by the OAR scheduler. For this work, we propose a deterministic expression for the buffer b that provides an approximate description of the scheduler's behavior, based on the constraints:
  - $0 \leq r \leq r_{max}$ (i.e. it is not possible to use more resources than those available in the cluster).
  - $0 \leq b \leq q$ (i.e. it is not possible to allocate more jobs that those available in the waiting queue).
- **Processing rate** p: CiGri's throughput is given by the number of finishing jobs in the cluster over a period of time Δt. In previous experiments [8], we could verify an approximately linear relationship between the throughput and the amount of running jobs. Based on this results, the throughput can be expressed as $p \times r$, where p corresponds to a single job's processing rate.

2) Dynamical model: Based on queuing theory, the dynamics can be described by the following system of difference equations,

$$\begin{cases} q_{k+\Delta t} = q_k + u_k - b_k \\ r_{k+\Delta t} = r_k + b_k - (p\Delta t) r_k \end{cases} \tag{1}$$

being q_k, r_k, u_k and b_k the above defined quantities at time k. The model is based on the interpretation that the amount of running jobs r behaves also as a queue, where its inflow is the outflow of the waiting queue q. The buffer function b_k is defined as,

$$b_k = \begin{cases} K(r_{max} - r_k)\Delta t & \text{if } q_k \geq K(r_{max} - r_k)\Delta t \\ q_k & \text{if } q_k < K(r_{max} - r_k)\Delta t \end{cases} \tag{2}$$

The behavior of the scheduler is modeled as a constrained proportional feedback loop that regulates the amount of running jobs r_k to an r_{max} reference value, where the gain K depends on the scheduling time. For the implementation of the control algorithm, and under the assumption that the scheduler is sufficiently fast, we approximate $K\Delta t \approx 1$.

The proposed model was then validated by comparing its response to that of the real system, using in both cases the same randomly generated input (Fig. 3).

[2]given by the amount of idle resources in the cluster and workload specific requirements: when idle resources in a cluster do not match hardware requirements specified by the user of a CiGri campaign (e.g. number of cores, amount of memory, etc.), they are considered not available

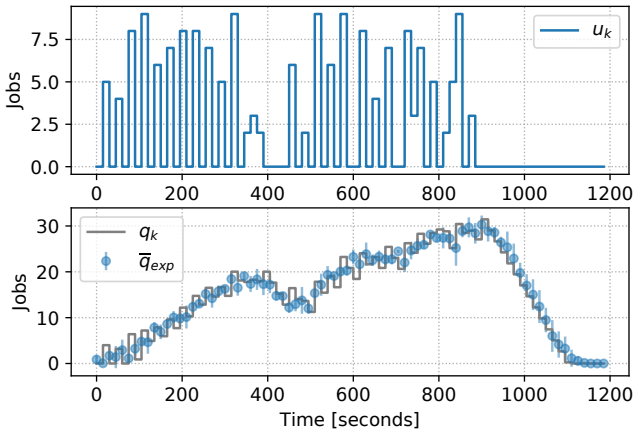


Fig. 3: Validation of the queue model against 10 averaged experiments performed in the real system ($\overline{q}_{exp}$).

B. File system analysis

To simplify the analysis, we focus on a specific kind of best-effort computing job that can be broken down to 3 stages: **Reading**: the job reads input data, **Processing**: the job performs certain computations over the input data, **Writing**: the job writes the result data.

To our knowledge, this structure represents the majority of the scientific workflows currently being processed by CiGri. While most of the time is generally spent on the processing stage (2nd step), the fileserver overloading comes from the reading and writing stages. It is noteworthy that this scheme defines an internal state of the job, which cannot be observed by CiGri.

Therefore, we propose to model the fileserver's load as a subsystem state $f_k \geq 0$, with two inputs u_k^1 and u_k^2 representing the contributions from the *reading* and *writing* steps of each job resp. Preliminary experiments showed that the dynamics of the system can be approximated by a first-order linear equation,

$$f_{k+\Delta t} = (1 - \alpha\Delta t) f_k + u_k^1 \Delta t + u_k^2 \Delta t.$$

The parameter α describes the *unloading* speed of the fileserver, which depends mainly on the averaging of the *loadavg* (load average) sensor of Unix-based systems. In this work, we focused on the impact of the *reading* step onto the fileserver (this implies $u_k^2 = 0, \forall k > 0$), that we model in terms of the number of allocations b, as it reflects the amount of starting jobs at each time instant,

$$f_{k+\Delta t} = (1 - \alpha\Delta t) f_k + k_{in} (b_{k-i})^\beta \Delta t,$$

which characterizes the type of loading by a gain k_{in}, a delay i and an exponent β. We performed several experiments with varying conditions so as to assess the impact on the parameters: filesize of 1 mb, 25 mb, 100 mb and 500 mb; hardware of the fileserver with different cores, and job average duration of 30 s, 15 min and 1 hr. Throughout the experiments, we could verify that the values of β, α and i remained constant, while the gain k_{in} varied considerably. Using a simple non-linear least squares algorithm, we obtained that $\beta \approx 2.05$, $\alpha \approx 0.015$, and $i = 2$ when fixing the cycle step $\Delta t = 5s$.

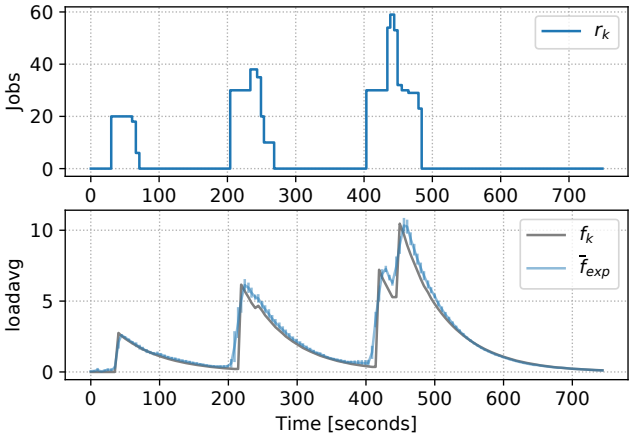


Fig. 4: Validation of the fileserver model against averaged experiments performed in the real system ($\overline{f}_{exp}$).

Parameter k_{in} depends strongly on the amount of files (and their size) used by each job, and so it varies for every different workflow. For this reason, a robust approach requires this parameter to be identified at runtime.

Following the same methodology previously used, we validated the proposed model against experiments not used in the modeling steps. In these cases, the cluster was composed of 60 resources, as shown in Figure 4.

C. Parameter estimation

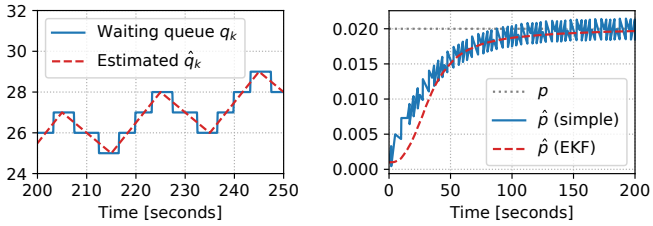
We define a CiGri environment, 1 cluster and M fileservers, and incorporate the parameters (the rate p and the gains $k_{in}^1, \ldots, k_{in}^M$) to be estimated in the state space formulation [18],

$$\begin{cases} q_{k+\Delta t} = q_k + u_k - b_k \\ r_{k+\Delta t} = r_k + b_k - (p_k \Delta t) r_k \\ f_{k+\Delta t}^1 = (1 - \alpha\Delta t) f_k^1 + k_{in}^1 (b_{k-i})^\beta \Delta t + w_k^1 \\ \quad \vdots \\ f_{k+\Delta t}^M = (1 - \alpha\Delta t) f_k^M + k_{in}^M (b_{k-i})^\beta \Delta t + w_k^M \\ p_{k+\Delta t} = p_k + w_k^{M+1} \\ k_{in,k+\Delta t}^1 = k_{in,k}^1 + w_k^{M+2} \\ \quad \vdots \\ k_{in,k+\Delta t}^M = k_{in,k}^M + w_k^{2M+1} \end{cases}$$

where $w_k^1, \ldots, w_k^M$ are the process noises associated to the impact of external agents on the storage system: in the infrastructure, fileservers are not dedicated to the CiGri environment, but shared among users that consume storage resources. The process noise $w_k^{M+1}, \ldots, w_k^{2M+1}$ represents the time-varying nature of the parameters in the system. For simplification, the experimental phase of this work was performed with two fileservers ($M = 2$).

The method used for the estimation process is the well-known EKF (Extended Kalman Filter), for being

able to deal with the non-linearities of the extended state-space representation. In accordance with the fluid modeling approach, we assume that the measurements of q and r (integer-valued states) are actually quantized measurements of the real ones (real-valued states) (Figure 5a). The filtering scheme contemplates this quantization effect as zero-mean measurement noise in both states. The reason behind this decision is that, in practice, simpler recursive parameter estimation algorithms showed quite oscillatory results due to the integer-valued nature of the state variables. This can be better seen in Figure 5b, where we simulated and compared both algorithms to illustrate this effect.



(a) Real-valued estimation $\hat{q}_k$ of integer variable q_k

(b) Simple (non-linear recursive leasts-squares) algorithm and EKF

Fig. 5: EKF implementation in fluid modeling scheme.

D. Control scheme

For the control loop, we implemented a model-predictive controller, mainly for the advantages that such an approach can deliver:

- It can easily deal with the constraints and non-linearities of the model.
- The model used for prediction is updated at each step with the parameters estimated by the EKF, making the scheme inherently adaptive to this variations in the model.
- It can cope with multiple objectives, as desired here.

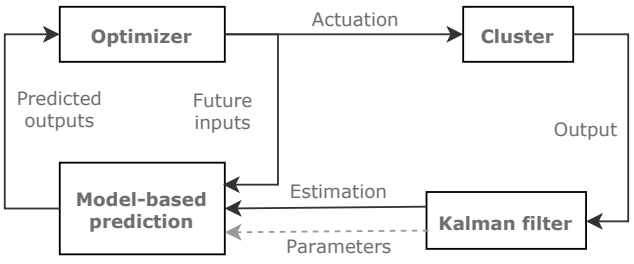


Fig. 6: Control scheme

Figure 6 illustrates the structure of the controller. The optimization process was simulated in Python using the constrained non-linear optimization library from SciPy, and then implemented in Ruby with the Ruby/GSL library for numerical computing.

As for the model-predictive scheme, a constant parametrization in the input was enforced, to simplify the optimization problem. Additionally, the prediction horizon was fixed to $N = 3$ as it gave acceptable results in the experiments.

Two different objective functions were formulated:

- **Maximize cluster usage**: maximizing the cluster's usage can be accomplished by guaranteeing enough waiting jobs q at every instant, which can be achieved by tracking a reference value q_{des},

$$J_k^1(u) = \sum_{j=0}^{N} \left(q_{k+j|k} - q_{des}\right)^2.$$

  It should be noted that maximizing the cluster usage by sending a maximal amount of jobs u is not an optimal strategy in practice, as it overloads the scheduler during execution.
- **Avoid fileserver overloading**: a system is overloaded if the *loadavg* is higher than the number of cores in the system (a rule of thumb widely used by system's administrators). Under this criteria, the value of the reference is dynamically adjusted to match the number of cores in the fileserver,

$$J_k^2(u) = \sum_{l=1}^{M} \sum_{j=0}^{N} \left(f_{k+j|k}^l - f_{des}^l\right)^2$$

where M is the number of file servers in the infrastructure and f_{des}^l the number of cores in file server l.

IV. Experimental results

A. Maximizing cluster usage

We performed 10 similar experiments in a cluster composed of 12 resources, in which we compared results when applying the naif approach and the MPC controller onto the waiting queue q when launching a 400 jobs campaign. In every case, the average cluster usage was computed as,

$$Usage[\%] = \frac{1}{t_f - t_0} \sum_{k=t_0}^{t_f} \frac{r_k \Delta t}{r_{max}},$$

with t_0 and t_f the initial and final time of the campaign execution respectively.

For every experiment, the reference of waiting jobs was set to 40 as it proved to be a value that achieved maximum cluster usage without producing significant stress on the scheduler (results in Figure 7). The original algorithm yields an average cluster utilization of 77% while the closed loop one raises it to 85%, showing an improvement of 8 percentage points, which achieves a reduction in the total computation time of 132 s.

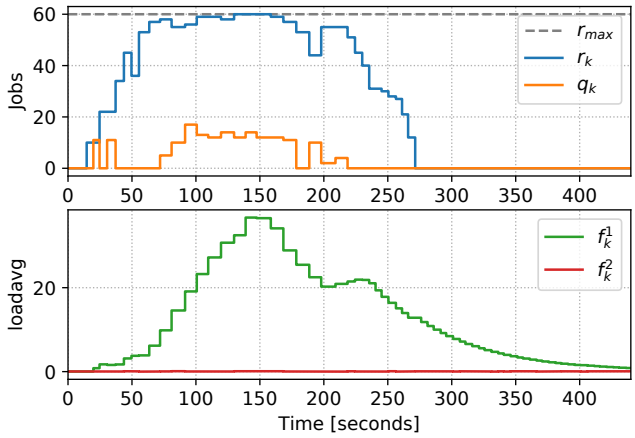
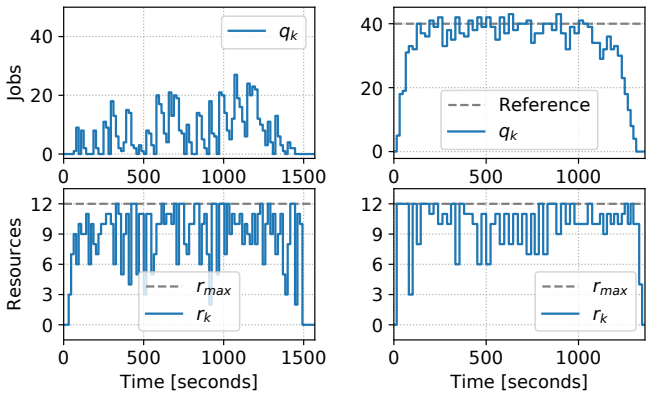


Fig. 8: Maximizing the used resources r_k yields an overload in the first fileserver (with load f_k^1) while it doesn't affect the second fileserver (with load f_k^2).



(a) (b)

Fig. 7: Original job submission policy (a) and closed loop behavior (b) when maximizing the amount of running jobs r_k to the number of available resources r_{max}.

B. Maximizing cluster while Controlling fileserver usage

In order to test the fileserver usage strategy, we performed a number of experiments on an infrastructure composed of a 60 resources cluster, and 2 fileservers with 4 cores each (which, as previously expressed, are assumed to be overloaded when $f_k > 4$). Then, we applied the cluster usage maximization strategy when computing a test workflow with average duration of 60 seconds. Each job in the workflow reads a file of 2 mb from the first fileserver, while the second fileserver is left unused. The results are as expected: there is a major impact on the fileserver load, that reaches 30, surpassing the overload value $f_k = 4$ (Figure 8).

We subsequently implemented the control loop for avoiding fileserver overload, that contemplates both loads f_k^1 and f_k^2. The results of the experiment can be seen in Figure 9, which shows how the controller is able to keep the load f_k^1 under the desired value. As a result,

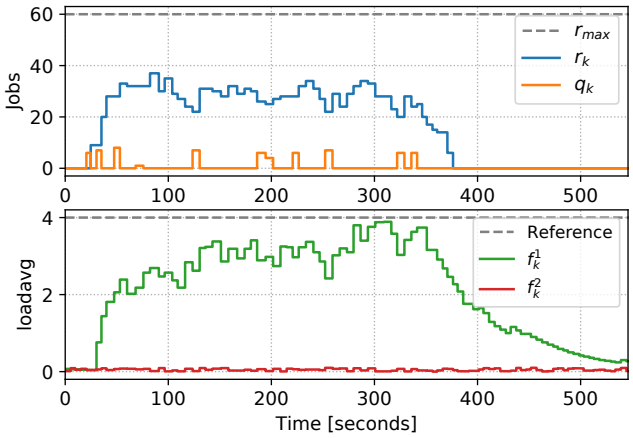


Fig. 9: Controlling load of both fileservers f_k^1 and f_k^2.

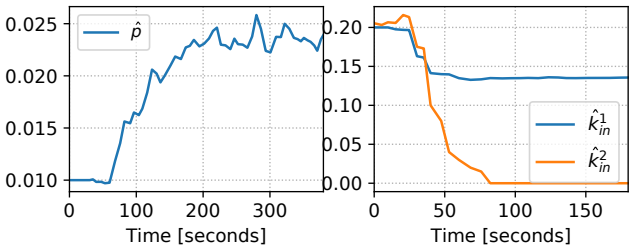


Fig. 10: Online estimation of $\hat{p}$, $\hat{k}_{in}^1$ and $\hat{k}_{in}^2$ using EKF.

it reduces considerably the cluster usage to around 45% (computed from the average of all 10 experiments) and, conversely, increases the total computation time of the campaign from 250 s in the first experiment to 345 s.

It is noteworthy that reducing the average cluster usage in around 50% achieves a reduction in the peak of the fileserver's load of 9 times between experiments (from 36.71 in the first experiment to 4 in the second one). Moreover, the EKF is able to estimate the processing rate p, and the gains k_{in}^1 and k_{in}^2, that reflects the impact of the workflow onto the fileservers, as shown in Figure 10. Given that the second fileserver is not used by the campaign, the associated gain k_{in}^2 remains null throughout the execution.

V. Discussion

A. Conclusion

In this work, we described results addressing administration in HPC systems through a control theory approach. First we performed an extensive analysis of the system and proposed a deterministic dynamical model. In opposition to the already analyzed previous works where black-box linear models were obtained based on classical identification tools, we propose a system described by parameters meaningful for computer scientists, which are related to the key aspects of the infrastructure. As a result, we achieved a model that

contemplates the varying behaviors of the real computing system.

Later on, we proposed a simple model-predictive control loop to achieve two different objectives: a) maximize cluster utilization by best-effort jobs, and b) control the fileserver's load due to the impact of the jobs. The accuracy of the prediction relies on a parameter estimation scheme based on the well-known EKF (Extended Kalman Filter) to adjust the model used for prediction to the real system, making the approach *adaptive* to parametric variations in the infrastructure.

The closed-loop approach achieved an average increase of 8 percentage points in performance when maximizing the cluster usage, and managed to avoid overloading the fileserver by deliberately restraining the submission of jobs to the grid when required.

While all the analysis and experiments were performed over the CiGri software, the problem is addressed in a general way, to allow the implementation on similar HPC computing platforms, as well as scalability to different infrastructures. Additionally, the solution is intended to be non-intrusive, as it doesn't interfere with the scheduling process.

B. Perspectives

While this work showed first results in the topic, there are still several aspects to be covered. For a start, we worked under the assumption that the scheduler is not overloaded during execution, and so its *loadavg* is not considered in the administration loop. CiGri deals with this cases by merely stopping job submission to an overloaded cluster, which acts as an *on-off* controller, and overrides the proposed control loop. A dynamical system's perspective can contribute to a much more efficient strategy in this regard.

Another pending issue is the file writing stage at the end of each job, usually a major cause of fileserver overloading, on which we are currently working.

The parameter estimation scheme proved to work on this particular problem, but we believe the strategy can be improved by studying more in depth the measurement and process noises involved in the EKF scheme, so that a more general framework can be achieved. Alternatively, other methods can be explored and compared with the current performance.

Finally, this very first approach was put to the test in an experimental setup rather simple. In order to validate our solution, further testing on different scenarios is required: with more diverse workloads, and more realistic resource usage by external users. The experiments were also limited in terms of scale: to our knowledge, some features and particularities of the system's behavior appear in larger configurations (i.e. the number of resources and running jobs in the order of thousands), and so this would be another interesting direction to explore.

References

[1] M. Berekmeri, D. Serrano, S. Bouchenak, N. Marchand, and B. Robu, "Feedback autonomic provisioning for guaranteeing performance in mapreduce systems," *IEEE Transactions on Cloud Computing*, vol. 6, no. 4, 2018.

[2] J. O. Kephart and D. M. Chess, "The vision of autonomic computing," *Computer*, vol. 36, no. 1, pp. 41–50, 2003.

[3] B. Khusainov, E. Kerrigan, A. Suardi, and G. Constantinides, "Nonlinear predictive control on a heterogeneous computing platform," pp. 11877–11882, ELSEVIER SCIENCE BV, 2017.

[4] N. Zhou, G. Delaval, B. Robu, E. Rutten, and J.-F. Méhaut, "An autonomic-computing approach on mapping threads to multi-cores fo software transactional memory," *Concurrency and Computation: Practice and Experience*, vol. 30, no. 18, 2018.

[5] E. Rutten, N. Marchand, and D. Simon, "Feedback Control as MAPE-K loop in Autonomic Computing," in *Software Engineering for Self-Adaptive Systems III*, vol. 9640 of *LNCS*, pp. 349–373, Springer, Jan. 2018.

[6] S.-M. Park and M. Humphrey, "Predictable high-performance computing using feedback control and admission control," *IEEE Trans. Parallel and Distributed Systems*, vol. 22, no. 3, 2011.

[7] H. A. Ghazzawi, I. Bate, and L. S. Indrusiak, "MPC vs. PID controllers in Multi-CPU multi-objective real-time scheduling systems," in *Proc. UK Electronics Forum*, pp. 77–83, 2012.

[8] E. Stahl, A. G. Yabo, O. Richard, B. Bzeznik, B. Robu, and E. Rutten, "Towards a control-theory approach for minimizing unused grid resources," in *Proc. 1st Int. Workshop on Autonomous Infrastructure for Science*, AI-Science'18, 2018.

[9] L. Malrait, S. Bouchenak, and N. Marchand, "Experience with ConSer: A system for server control through fluid modeling," *IEEE Transactions on Computers*, vol. 60, no. 7, pp. 951–963, 2011.

[10] C. Biscarat and B. Bzeznik, "Synergy between the CIMENT tier-2 HPC centre and the HEP community at LPSC in Grenoble (France)," in *Journal of Physics: Conference Series*, vol. 513, p. 032008, IOP Publishing, 2014.

[11] N. Capit, G. Da Costa, Y. Georgiou, G. Huard, C. Martin, G. Mounié, P. Neyron, and O. Richard, "A batch scheduler with high level components," in *IEEE Int. Symp. on Cluster Computing and the Grid, CCGrid*, pp. 776–783, 2005.

[12] R. Bendel, S. Higgins, J. Teberg, and D. Pyke, "Comparison of skewness coefficient, coefficient of variation, and gini coefficient as inequality measures within populations," *Oecologia*, vol. 78, no. 3, pp. 394–400, 1989.

[13] M. Berekmeri, D. Serrano, S. Bouchenak, N. Marchand, and B. Robu, "A control approach for performance of big data systems," *IFAC Proceedings Volumes*, vol. 47, no. 3, pp. 152–157, 2014.

[14] L. Baresi, S. Guinea, A. Leva, and G. Quattrocchi, "A discrete-time feedback controller for containerized cloud applications," in *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, pp. 217–228, ACM, 2016.

[15] E. B. Lakew, A. V. Papadopoulos, M. Maggio, C. Klein, and E. Elmroth, "Kpi-agnostic control for fine-grained vertical elasticity," in *Proceedings of the 17th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*, pp. 589–598, IEEE Press, 2017.

[16] Y. Georgiou, O. Richard, and N. Capit, "Evaluations of the lightweight grid cigri upon the grid5000 platform," in *IEEE Int. Conf. on e-Science and Grid Computing*, 2007.

[17] R. Jain, J. Werth, and J. C. Browne, *Input/output in parallel and distributed computer systems*, vol. 362. Springer Science & Business Media, 2012.

[18] E. A. Wan and R. Van Der Merwe, "The unscented kalman filter for nonlinear estimation," in *Adaptive Systems for Signal Processing, Communications, and Control Symposium 2000. AS-SPCC*, pp. 153–158, 2000.